A Python Library with Fast Algorithms for Popular Entropy Definitions

George Manis[1] and Roberto Sassi[2]

[1] Department of Computer Science and Engineering,
School of Engineering, University of Ioannina, Ioannina, Greece
[2] Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy

Abstract

We present "fastEntropyLib", a set of Python functions, organized as a library, to facilitate the computation of popular definitions of entropy, used in biomedicine. Entropy estimation is a powerful tool in non-linear analysis of time series, especially for heart rate signals and electroencephalograms. However, the computation of the most common entropy metrics is time consuming, an inhibitor, especially when large amount of data is to be processed.

The library includes the most common definitions of entropy: Shannon Entropy, Rényi Entropy, Approximate Entropy, Sample Entropy, as well as the more recently proposed Bubble Entropy, an entropy almost free of parameters. What makes this library different from other similar Python libraries is the employment of fast algorithms for the implementation of Approximate, Sample Entropy and Bubble Entropy. For both Approximate and Sample Entropy, the Bucket Assisted and Lightweight algorithms have been employed. These two algorithms speed up the computation significantly by excluding similarity comparisons, which we know in advance they will fail. This is the first time code for these two algorithms becomes publicly available. It is also the first time that fast code becomes publicly available for Bubble Entropy. Even though bubble sort is a quadratic algorithm, the computation of Bubble Entropy is done in linear time, exploiting, in each step, already sorted vectors.

Since speed is the weak point of Python, we selected to implement all algorithms in C and add a Python wrapper on top of them. In this way, we exploit both the speed of C and the convenience and popularity of Python.

1. Introduction

With the term *entropy* in thermodynamics we express the disorder of a system. In information theory, entropy is a measure of the amount of information in a message. In time series analysis, entropy is a measure of complexity or irregularity. Entropy expresses the mean uncertainty, or in other words, how many bits we need to describe a system. The most popular and widely used definition of Entropy was proposed by Shannon. Rényi generalized this definition. Thousands of papers were based on Pincus definition called *Approximate Entropy* proposed in 1991 [1, 2]. Approximate Entropy played a significant role in making entropy so popular in biomedical time series analysis. Some years later, Richman and Moorman proposed *Sample Entropy* [3, 4], suggesting simple but important modifications on Approximate Entropy. Sample Entropy became even more popular than Approximate Entropy and seems to be a golden standard today.

The main drawback of both Approximate and Sample Entropy is the increased computational time necessary to run the algorithms. In [5] and [6] fast algorithms have been proposed, which speed up the computation of those method significantly. Code for these implementations was not been published until today. In the proposed library, algorithms are implemented according to the description in [5] for Approximate Entropy and in [6] for Sample Entropy.

Other definitions of entropy have also appeared trying to give additional information on the disorder of the system. Some of them tried to improve the existing ones, some others tried to suggest more novel ideas. Included in this library is *Bubble Entropy* [7], an entropy free of parameters.

The importance of this paper can be summarized as follows: (a) it introduces and gives details on a set of functions, organized as a library, computing popular definitions of entropy; the programmer can use an integrated set of tools and compute entropy based on all definitions in a simple, uniform way, (b) both Approximate Entropy and Sample Entropy are computed using fast algorithms written in C; this is the first time these implementations [6] become publicly available, (c) it is also the first time that code becomes available for Bubble Entropy; A fast algorithm is also used based on the description of the algorithm in [7].

The library is publicly available and free to use. Please use the appropriate references. The link where someone can download the library is the following:

`http://www.cs.uoi.gr/~manis/pythonFastEntropy`

The rest of the paper is structured as follows. Section 2 summarizes similar libraries, available today in Python, focusing on estimation of entropy definitions. A detailed description of the proposed library follows, in section 3. Section 4 summarizes this work.

2. Entropy Libraries in Python

Python is the most rapidly developed programming language today. Python is a language easy for someone to learn and use and also fast in code development. The large number of libraries available can further speed up development and alleviate the programmers' task. The amount of ready to use code increases day by day.

Libraries with functions for the estimation of entropy, as well as other tools for non-linear analysis, can be easily found on the internet. However, there is still much work which should be done towards a complete library which can be accepted as a standard. In the following, some of the most important Python libraries providing functions for entropy analysis are outlined.

Dit [8] is a Python package for information theory. It provides functions for the computation of Shannon Entropy, Rényi Entropy, Tsallis Entropy, Necessary Conditional Entropy, Residual Entropy. It also provides other tools for the estimation of Independent Information, Variation of Information, Mutual Information and Divergences

Nolds [9] is a small numpy-based library that provides an implementation and a learning resource for nonlinear measures for dynamical systems based on one-dimensional time series. Currently the following measures are implemented: Sample Entropy, Correlation Dimension, Lyapunov Exponent, Hurst Exponent, Detrended Fluctuation Analysis.

PyEntropy [10] is a small set of functions on top of NumPy for different definitions of entropy. Currently available are: Shannon Entropy, Sample Entropy, Multiscale Entropy, Composite Multiscale Entropy, Permutation Entropy, Multiscale Permutation Entropy.

The motivation to propose a new entropy library in Python is strong. Even though there is a number of Python libraries already available for the computation of several entropy definitions, this new library focuses on smart implementation of entropy algorithms, ensuring low execution times, alleviating the most important drawback of entropy computation.

3. Description of *fastEntropyLib*

fastEntropyLib consists of a set of Python functions, each corresponding to one of the above discussed definitions. We will present, in the following, the interface for each one of those functions, as well as some details on their implementation, where this is interesting.

— `shannon_entropy (data, bin_size=1)`

Function for the computation of Shannon Entropy. $Data$ is the input on which the entropy will be computed, usually a time series. Bin_size is the size of the bins for the histogram. The first bin starts from the minimum element of the data.

— `renyi_entropy(data, order=2, bin_size=1)`

This function is for the computation of Rényi Entropy. The parameters are the same with those of Shannon Entropy, but there is also an additional parameter, the $order$, for the order of Rényi Entropy. The default value is $order = 2$, since this is the most common value used.

— `sample_entropy (timeseries, m=2, r=0.2, algorithm='bucket', bucket_split=5)`

Function for the computation of Sample Entropy. The first parameter is again the parameter for the input of the examined data, a time series here. We gave it the name $timeseries$ this time, since Sample Entropy is more meaningful for time series.

The computation of Sample Entropy request the estimation of two parameters, the embedding dimension m and the threshold r. The selection is a critical point in the application of the method, since these parameters do not only influence the computed value, but also affect the discriminating power. Typical values have been suggested and widely used to alleviate the problem and standardized the application of the method. The function accepts the parameters m and r with default values $m = 2$ and $r = 0.2$ according to the common practice.

We provide three different implementations for the computation of the method. All three return exactly the same value, none of them is based on estimations. The programmer can use the parameter $algorithm$ and define which implementation to select. The three options are: *straightforward, bucket* and *lightweight*. The option *straightforward* calls an algorithm directly resulting from the definition of the method. The code has been written in Python. It is the slowest one, but fast enough for small time series. It is a simple implementation, easy to be understood and has been included in the library for completeness and clarity. The default value for $algorithm$ is $algorithm =$ "$bucket$".

The main problems with Sample Entropy is the high computational complexity which, according to the definition of the method, is $O(n^2 m)$. This results in a demand of computational power, especially when a large number of time series is analyzed (which is usually the case). Two algorithms has been proposed to speed up the computation of the method, something that they achieve in a remarkable degree [6]. The implementation of both algorithms has been done in C (wrapped up with a Python interface)

something that makes their execution even faster. A short description of the two algorithms follows, but for more details please see [6].

The main idea of the Bucket Assisted algorithm is to detect early and exclude comparisons, which we know that they will fail. Such comparisons are those that the sums of the elements of the two vectors differ more than $m*r$. Thus, we put the vectors in buckets according to the sum of their elements and we compare only those buckets which we know that they possibly contain similar vectors. Two vectors cannot be similar if the distance between their buckets is more than $r*m$. Exploiting this property we can avoid a large number of comparisons. In addition, vectors inside the buckets are sorted according to their first elements, allowing exclusion of massive comparisons between vectors their first elements of which differ more than r. The gain in computation time is remarkable, exploiting these two ideas.

The Lightweight algorithm is a similar algorithm which exploits only the second idea. For specific values of m and r and time series lengths, the overhead to manage buckets is more than the gain we have. For those cases, the users may prefer to use the Lightweight algorithm, for even faster executions.

The Bucket Assisted algorithm allows the user to play with the parameter $bucket\_split$. This parameter can offer even better performance. Even thought the default value $bucket\_split = 5$ is not necessary to be modified and gives a good acceleration, the users can also play with other integer values of $bucket\_split$, if they wish.

Even though the gain of using these algorithms compared to the straightforward implementation depends on the compiler used, the bucket assisted and the lightweight algorithms are always much faster. implementation in C also speeds ups the execution significantly. For details on algorithm performance issues please see [6].

— `approximate_entropy (timeseries, m=2, r=0.2, algorithm='bucket', bucket_split=5)`

This is the function for the computation of Approximate Entropy. One can notice that the interface is identical with that of Sample Entropy. However, this is not a surprise, since the two methods have the same parameters and similar definitions. The Bucket Assisted algorithm have been implemented as it was described in [5], whilst the Lightweight algorithm is a modification of the algorithm presented in [6]. Implementations are similar to the implementations discussed earlier for Sample Entropy. For performance issues please see [5, 6].

— `bubble_entropy (timeseries, m)`

Function for the computation of Bubble Entropy. Again, the parameter $timeseries$ is for the input time series and m is the size of the embedding space. There is no default value for m, since no value has been suggested or practically used yet as typical. Suggested values for the user to try are $m \leq 20$.

The implementation is based on the description in [7]. The computation of Bubble Entropy requests (bubble) sorting of vectors of size m. With bubble sort, this costs $O(m^2)$ steps, with quick sort $O(mlogm)$. Sorting is a generally expensive task. However, the $N-m+1$ vectors, which are to be sorted, are not independent the one from the other. They are ordered and produced by the same time series. Each vector v_i in the embedding space has $m-1$ common elements with its preceding vector v_{i-1}. We exploit this property. When we want to sort the vector v_i, the preceding vector v_{i-1} is already sorted. In order to produce the sorted vector v_i, we take the already sorted vector v_{i-1} as a starting point. We remove from v_{i-1} the element which does not belong to v_i. This was the first element of v_{i-1}, before sorting it. This is done in $O(m)$ time. After removing this element, our vector consists of $m-1$ sorted elements. The missing element is the last element of v_i. We need to insert it in the correct position, so that the vector will remain sorted. This can be done also in $O(m)$ time, using insertion sort. The above algorithm can compute Bubble Entropy very fast, in linear time $O(nm)$.

4. Conclusions

In this paper, we presented a Python library which facilitates the user with functions for the computation of definitions of entropy. The library focuses on the most popular entropy measures used in biomedicine and specifically in biomedical time series analysis. The library provides functions for the computation of Shannon Entropy, Rényi Entropy, Approximate Entropy, Sample Entropy and Bubble Entropy. The code for the computationally intensive Approximate Entropy and Sample Entropy has been developed according to algorithms which allow fast computation. To further speed up the execution, the code for Approximate and Sample Entropy has been developed in C and is provided with a Python wrapper function. It is the first time that code for those algorithms becomes publicly available, not only for Python, but for any other language. It is also the first time that code become publicly available for Bubble Entropy. We believe that the proposed library will become a useful tool in biomedical time series analysis.

References

[1] Pincus SM. Approximate entropy as a measure of system complexity. Proc Natl Acad Sci March 1991;88(6):2297–2301.

[2] Pincus SM, Goldberger AL. Physiological time-series analysis: what does regularity quantify. Am J Physiol Heart Circ Physiol 1994;266:1643–1656.

[3] Richman JS, Moorman JR. Physiological time series analysis using approximate entropy and sample entropy. Am J Physiol Heart Circ Physiol 2000;278:2039–2049.
[4] Lake DE, Richman JS, Griffin MP, Moorman JR. Sample entropy analysis of neonatal heart rate variability. Am J Physiol Regul Integr Comp Physiol September 2002; 283(3):R789–R797.
[5] Manis G. Fast computation of approximate entropy. Comput Methods Programs Biomed 2008;91(1):48–54.
[6] Manis G, Md. Aktaruzzaman, Sassi R. Low computational cost for sample entropy. Entropy 2018;20(1):61.
[7] Manis G, Md. Aktaruzzaman, Sassi R. Bubble Entropy: an entropy almost free of parameters. Transactions on Biomedical Engineering November 2017;64(11):1558–2531.
[8] James RG, Ellison CJ, Crutchfield JP. dit: a Python package for discrete information theory. The Journal of Open Source Software 2018;3(25):738.
[9] Nolds, 2017. https://pypi.org/project/nolds/, Last released: Nov 30, 2017, Last accessed: Aug 5, 2021.
[10] pyEntropy. https://github.com/nikdon/pyEntropy, Last accessed: Aug 5, 2021.

Address for correspondence:

George Manis
Dept. of Computer Science and Engineering, School of Engineering, University of Ioannina, Ioannina 45110, Greece
manis@cs.uoi.gr

Roberto Sassi
Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy
roberto.sassi@unimi.it